Computer Graphics (UCS505)

Project on Snake Game

Submitted By

Satinder Kaur	101917001
Poorvika Khanna	101917021
Shreya R. Singh	101917031

B.E. Third Year – COPC

Submitted To:

Dr Samya Muhuri

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

Table of Contents

Introduction To Project

Mr. Severus is a famished snake who wanders around looking for food in order to survive. He eats a lot yet his favorites are fruits. Save him from reaching the boundary. And don’t let him consume himself. Save him and the snake lives.

The player controls the snake's movement in this project by altering its direction. Snake goes onward in the current direction automatically. To get a piece of fruit, the player must move through it while directing the snake's head to it. The snake will grow in size as it consumes the fruit.

The goal is to consume as many fruits as possible while staying within the boundaries, with the primary goal of surviving as long as possible. How long can you keep your tail from becoming your dinner?

Working/Explanation

The player will be ready to begin the game as soon as the window appears.

When the game begins, the Snake moves in a random direction in its regular set-size. A fruit appears on the screen at random, and the player must move the snake in all four directions to get it to eat the yellow squares (fruits).

- Up arrow key (For moving up)
- Left arrow key (For moving left)
- Down arrow key (For moving down)
- Right arrow key (For moving right)

After consuming each fruit, the snake's length grows by one unit, and it moves on to consume more fruits. The game can end in one of two ways:

- Snakes eats himself
- Snake reaches boundary

As a result of consuming the fruits, the snake's length grows, but if he eats himself or reaches the game's boundary, the game is over.

Computer Graphics concepts used

- **OpenGL** is an application programming interface for rendering 2D and 3D vector graphics that works across languages and platforms. To achieve hardware-accelerated rendering, the API is often used to interact with a graphics processing unit.

- The OpenGL Utility Toolkit (**GLUT**) is a toolkit for building OpenGL programmes that is independent of the window system. It creates a simple OpenGL windowing application programming interface (API).

- **glutInit()** is used to initialize the GLUT library.

- When creating a window, **glutInitDisplayMode()** is used to indicate the type of display mode. The color mode, quantity, and buffer type are all specified using mode. The default color mode is RGBA. The drawing instruction is executed in a buffer zone where the drawing is very quick, resulting in double buffering. Following the completion of the drawing instruction, the completed drawing will be displayed on the screen instantly via the exchange instruction, preventing incomplete drawings and increasing efficiency. It's commonly used to create animation effects.

- **glutMainLoop()** enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

- The **glClearColor()** function was used to change the background color.
  The red, green, blue, and alpha values used by glClear to clear the color buffers are specified by glClearColor. glClear, clear buffers to preset values.

- **glutDisplayFunc()** sets the current window's display callback. The display callback for the window is triggered when GLUT detects that the window's normal plane needs to be redisplayed.

- **glutReshapeFunc()** sets the reshape callback for the current window. When the window is first constructed, reshape callback is called, and then every time it is resized or moved after that. So we can effectively utilize this function to set up our viewport because we need to reset the viewport and projection of our application every time the window is created or resized.
- Since we need to alter the matrix mode to the projection matrix, **glMatrixMode** is used to set up the projection.

- The **glBegin** and **glEnd** subroutines delimit the vertices that define a primitive or group of like primitives. We used the **GL_LINE_LOOP** argument of the glBegin method to draw grid squares.
  To draw a primitive on the screen, we first call glBegin and provide the type of primitive we wish to draw in the mode parameter. The vertices we describe are connected by GL LINE LOOP, and a loop is produced. The significance of a loop of lines is that the given first and end vertices will be connected. As a result, a close poly will form.
  Then, after listing all vertices one by one, call glEnd to notify OpenGL that we've finished displaying a primitive.
  We used **glLineWidth** in the intermediate to specify the rasterized width of lines. Each vertex's color is determined by **glColor3f**. The **glVertex** function commands are used to provide point, line, and polygon vertices within glBegin/glEnd pairs. The number "2" in the name refers to the number of parameters supplied to the function.

- With the help of the **glcolor3f** function, we set a condition that if the coordinates are of boundary, set the colors and line width of the grid to be different from the one with the color in between.

- To fill various polygons like snake and fruit we used the glColor3f function which sets the respective colors.

- At the end of the display callback, we swap the buffers, resulting in a new frame being displayed, and our FPS is the number of times the display callback in our programme is called in one second. We're using it to move the snake around on the screen. Each time a new frame is displayed, we will move the snake one box forward.
  glutTimerFunc registers a timer callback to be triggered in a specified number of milliseconds.
  glutPostRedisplay marks the current window as needing to be redisplayed. It urges opengl to call the display function as quickly as possible. The display function will be called once the timer callback function is called. As a result, each

time the timer callback method is called, a new frame is displayed. This is because after a certain amount of milliseconds, a continuous loop of frames is presented. As a result, the snake will move, and the coordinates will be updated to match the snake's direction as indicated by the arrow keys on the keyboard.

- **glutSpecialFunc** sets the special keyboard callback for the current window. The special keyboard callback is triggered when keyboard function or directional keys are pressed.

User Defined Function

1. **initGrid():** First, a function will be created to set the coordinates of the grid that will be created. On a coordinate system, one unit of grid is equal to one unit.

2. **unit():** Set the location where the square will be drawn specified by the draw_grid() function.

3. **draw_grid():** Draws the whole grid every time a new frame is displayed.

4. **draw_snake():** Draws the snake body and modifies the movement coordinates based on the keys. And it's also responsible for the snake's length increasing after he eats the fruit. The game will be over once the snake crosses the boundary line.

5. **draw_food():** Draws the food at a random position every time the food is consumed by the snake.

6. **random():** The srand() function sets the seed for the rand() function. The result of a call to time(0) should be used as the seed. The number of seconds is returned by the time() function. As a result, seed value varies over time. As a result, each time we run the programme, we get a new set of random integers.

Source Code

- game.cpp

```
#include <GL/glut.h>
#include <iostream>
#include <ctime>
#include "game.h"

void unit(int,int);
int random(int,int);

bool length_inc=false;
bool seedflag = false;
extern int score;
extern bool game_over;
bool food=false;
int rows=0,columns=0;
int sDirection = RIGHT;
int foodx,foody;
int posx[MAX+1]={4,3,2,1,0,-1,-1};
int posy[MAX+1]={10,10,10,10,10,10,10};
int length=7;

void initGrid(int x,int y)
{
    columns=x;
    rows=y;
}

void draw_grid()
{
    for(int i =0;i<columns;i++)
    {
        for(int j=0;j<rows;j++)
            {unit(i,j);}
    }
}
```

```
void draw_snake()
{
    for(int i =length-1;i>0;i--)
    {
        posx[i]=posx[i-1];
        posy[i]=posy[i-1];
    }
    for(int i=0;i<length;i++)
    {
        glColor3f(0.0,1.0,0.0);
        if(i==0)
        {
            glColor3f(1.0,0.0,0.0);
            switch(sDirection)
            {
            case UP:
                posy[i]++;
                break;
            case DOWN:
                posy[i]--;
                break;
            case RIGHT:
                posx[i]++;
                break;
            case LEFT:
                posx[i]--;
                break;
            }
            if(posx[i]==0||posx[i]==columns-1||posy[i]==0||posy[i]==rows-1)
                game_over=true;
            else if(posx[i]==foodx && posy[i]==foody)
            {
                food=false;
                score++;
                if(length<=MAX)
                    length_inc=true;
            }
            for(int j=1;j<length;j++)
            {
                if(posx[j]==posx[0] && posy[j]==posy[0])
                    game_over=true;
            }
        }
```

```
        }
        glBegin(GL_QUADS);
        glVertex2d(posx[i],posy[i]);
        glVertex2d(posx[i]+1,posy[i]);
        glVertex2d(posx[i]+1,posy[i]+1);
        glVertex2d(posx[i],posy[i]+1);
        glEnd();
    }
    if(length_inc)
    {
        length++;
        length_inc=false;
    }
}

void draw_food()
{
    if(!food)
    {
        foodx=random(2,columns-2);
        foody=random(2,rows-2);
        std::cout<<foodx<<foody<<std::endl;
        food=true;
    }
    glBegin(GL_QUADS);
        glVertex2d(foodx,foody);
        glVertex2d(foodx+1,foody);
        glVertex2d(foodx+1,foody+1);
        glVertex2d(foodx,foody+1);
    glEnd();
}

void unit(int x,int y)
{
    glLoadIdentity();
    if(x==0||x==columns-1||y==0||y==rows-1)
    {
        glLineWidth(4.0);
        glColor3f(1.0,1.0,0.0);
    }
    else
    {
        glColor3f(1.0,1.0,1.0);
```

```
        glLineWidth(1.0);
    }
    glBegin(GL_LINES);
        glVertex2d(x,y); glVertex2d(x+1,y);
        glVertex2d(x+1,y); glVertex2d(x+1,y+1);
        glVertex2d(x+1,y+1); glVertex2d(x,y+1);
        glVertex2d(x,y+1); glVertex2d(x,y);
    glEnd();
}
int random(int _min,int _max)
{
    if(!seedflag)
    {
        srand(time(NULL));
        seedflag=true;
    }
    else
        seedflag=false;
    return _min+rand()%(_max-_min);
}
```

- game.h

```
#ifndef GAME_H_INCLUDED
#define GAME_H_INCLUDED

#define MAX 50
#define UP 1
#define RIGHT 2
#define DOWN -1
#define LEFT -2

void initGrid(int,int);
void draw_grid();
void draw_food();
void draw_snake();

#endif // GAME_H_INCLUDED
```

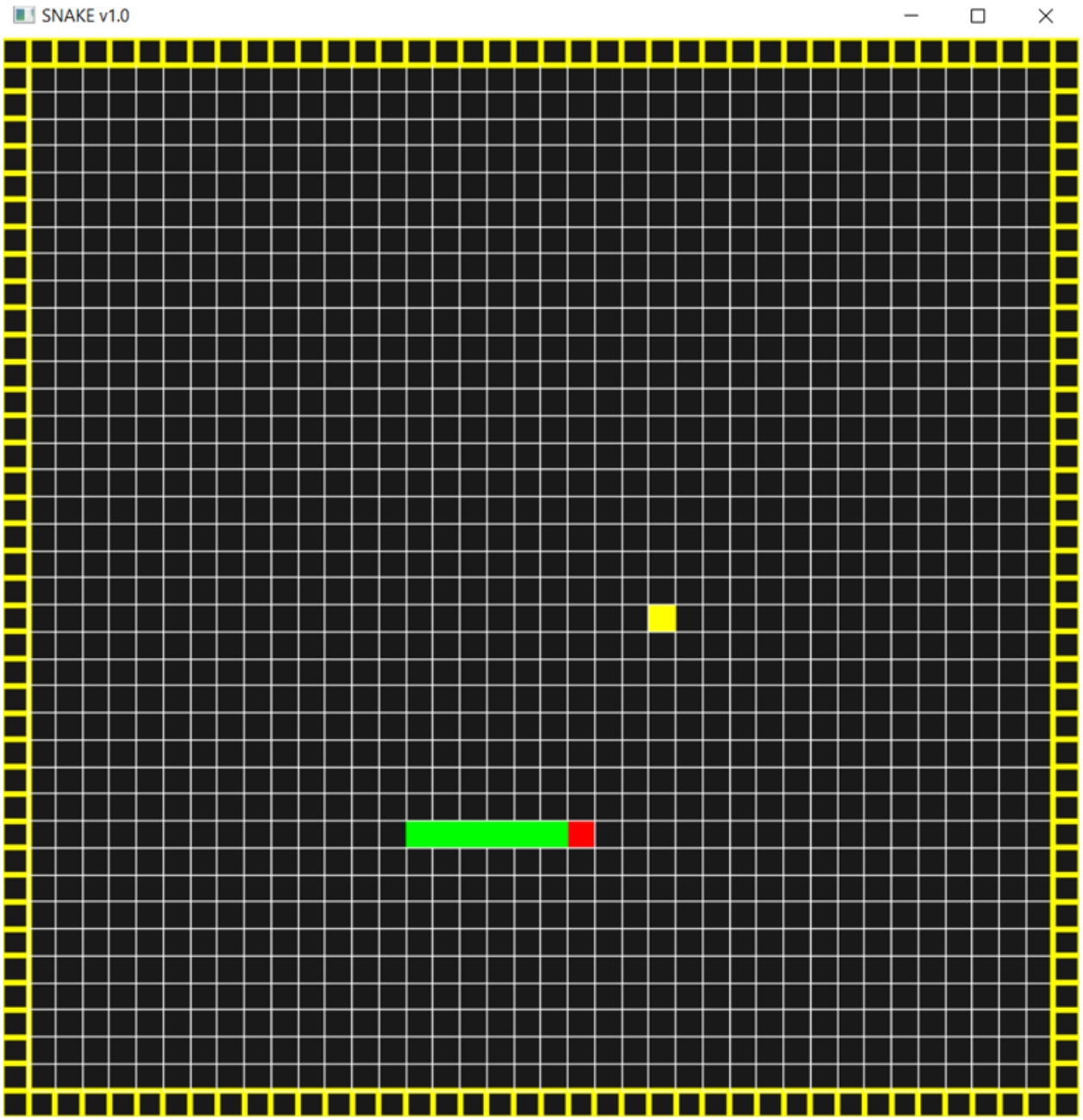
- main.cpp

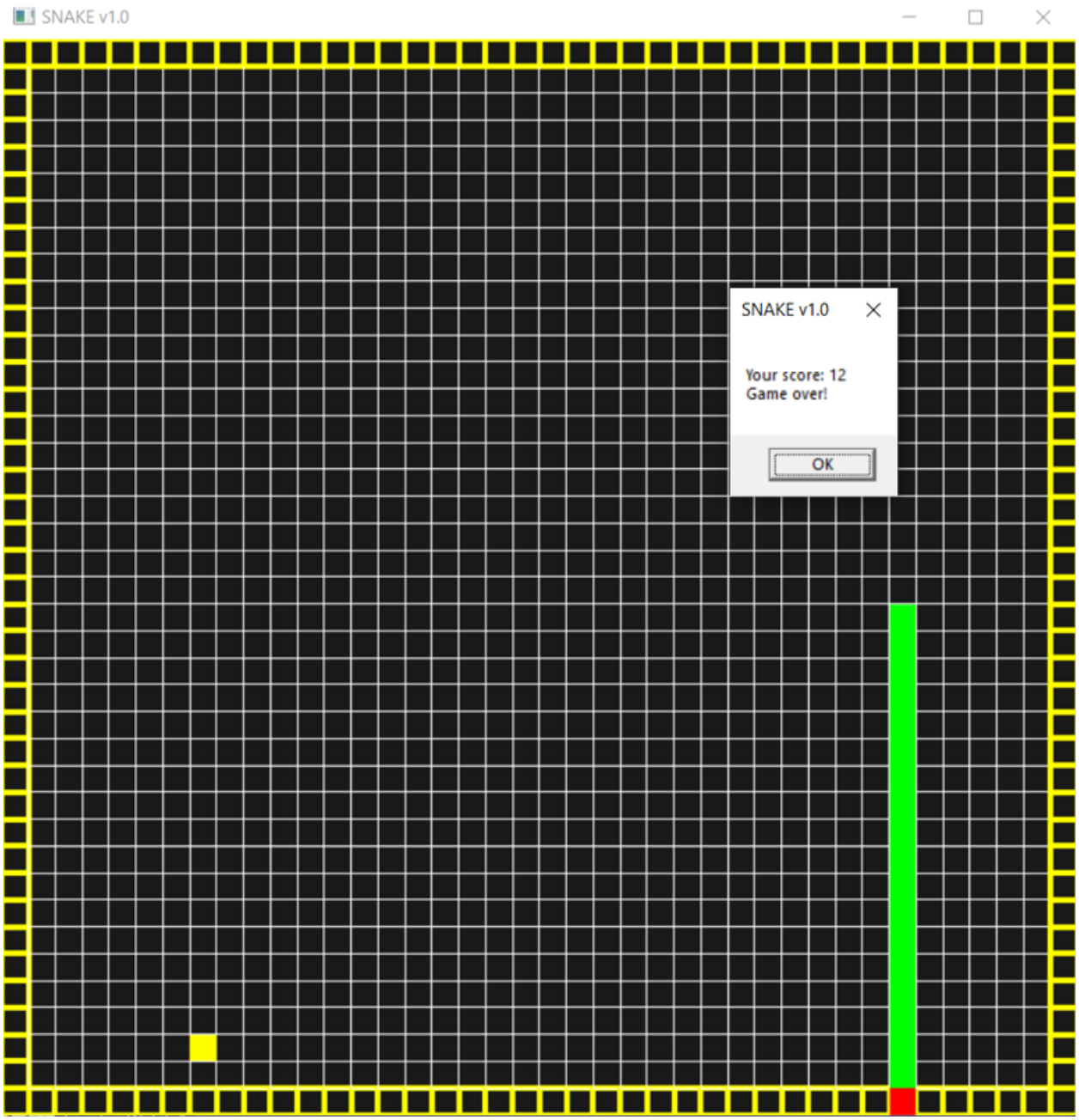
```
#include <GL/glut.h>
#include <iostream>
#include <fstream>
#include "game.h"
#include <sstream>
#include <windows.h>

#define ROWS 40.0
#define COLUMNS 40.0

std::ofstream ofile;
std::ifstream ifile;
bool game_over=false;
extern int sDirection;
int score=0;

void init();
void display_callback();
void input_callback(int,int,int);
void reshape_callback(int,int);
void timer_callback(int);

int main(int argc,char**argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutInitWindowPosition(10,10);
    glutInitWindowSize(750,750);
    glutCreateWindow("SNAKE v1.0");
    glutDisplayFunc(display_callback);
    glutReshapeFunc(reshape_callback);
    glutSpecialFunc(input_callback);
    glutTimerFunc(100,timer_callback,0);
    init();
    glutMainLoop();
    return 0;
}
```

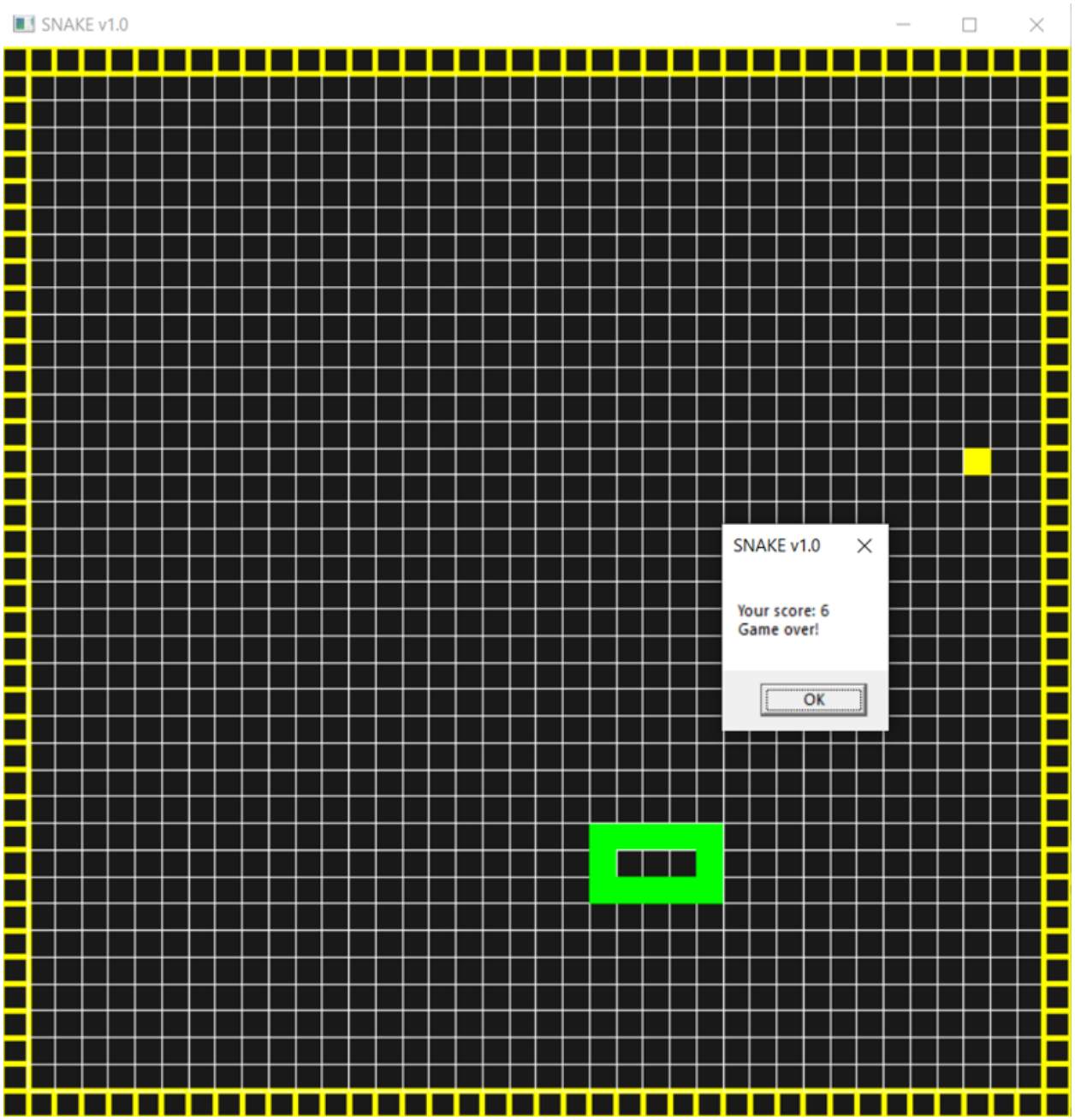
```
void init()
{
    glClearColor(0.1,0.1,0.1,0.1);
    initGrid(COLUMNS,ROWS);
}

//Callbacks
void display_callback()
{
    if (game_over)
    {
        ofile.open("score.dat", std::ios::trunc);
        ofile << score << std::endl;
        ofile.close();
        ifile.open("score.dat", std::ios::in);
        char a[4];
        ifile >> a;
        std::cout << a;
        char text[50] = "Your score : ";
        strcat(text, a);
        std::stringstream box_message;
        box_message << "Your score: " << score<<"\nGame over!";
        MessageBoxA(0, box_message.str().c_str(), "SNAKE v1.0", MB_OK);
        exit(0);
    }
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    draw_grid();
    draw_food();
    draw_snake();
    glutSwapBuffers();
}
void reshape_callback(int w, int h)
{
    glViewport(0,0,(GLfloat)w,GLfloat(h));
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0,COLUMNS,0.0,ROWS,-1.0,1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

```
void timer_callback(int)
{
    glutPostRedisplay();
    glutTimerFunc(100,timer_callback,0);
}
void input_callback(int key,int x,int y)
{
    switch(key)
    {
    case GLUT_KEY_UP:
        if(sDirection!=DOWN)
            sDirection=UP;
        break;
    case GLUT_KEY_DOWN:
        if(sDirection!=UP)
            sDirection=DOWN;
        break;
    case GLUT_KEY_RIGHT:
        if(sDirection!=LEFT)
            sDirection=RIGHT;
        break;
    case GLUT_KEY_LEFT:
        if(sDirection!=RIGHT)
            sDirection=LEFT;
        break;
    }
}
```

Game Screenshots




SNAKE v1.0
SNAKE v1.0
Your score: 12
Game over!
OK


SNAKE v1.0
SNAKE v1.0
Your score: 6
Game over!
OK